Efficient Trace for RISC-V: Design, Evaluation, and Integration in CVA6

Umberto Laghi, Simone Manoni, Emanuele Parisi, Andrea Bartolini

Department of Electrical, Electronic, and Information Engineering (DEI) – University of Bologna, Italy

Abstract

In this work, we present the design and evaluation of a Processor Tracing System compliant with the RISC-V Efficient Trace specification for Instruction Branch Tracing. We integrate our system into the host domain of a state-of-the-art edge architecture based on CVA6. The proposed Tracing System introduces a total overhead of 9.2% in terms of resource utilization on a Xilinx VCU118 FPGA on the CVA6 subsystem while achieving an average compression rate of 95.1% on platform-specific tests, compared to tracing each full opcode instruction.

Introduction

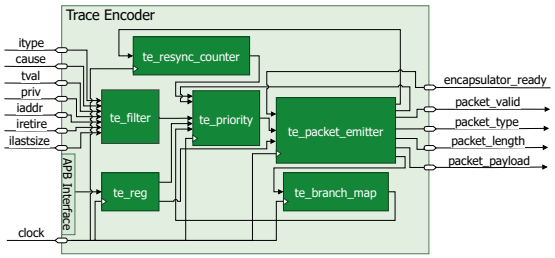
In modern computing systems, understanding program execution is challenging, as software behavior and performance can deviate from expectations due to interactions with other cores, real-time events, suboptimal software implementations, or a combination of these factors. Profiling techniques commonly used to analyze code and monitor program execution often involve significant trade-offs between being intrusive, enabling detailed debugging, or providing only high-level insight into execution performance [1].

Instruction Branch Tracing (IBT) allows continuous, non-intrusive, fine-grained monitoring of program execution by tracking program counter (PC) address deltas induced by *special* instructions: jump, call, return, branch, interrupts, or exceptions. RISC-V (RV) provides a standardized IBT mechanism known as Efficient Trace (E-Trace) [2]. It splits the code into blocks, where each block is an instruction sequence bounded by two special instructions, whenever a discontinuity occurs, a dedicated hardware *Trace Encoder* (TE) module generates a trace packet. The trace packet sequence is then processed by a software *Trace Decoder* (TD) running on a host machine, which reconstructs the program execution by integrating the trace packet data with the program binary.

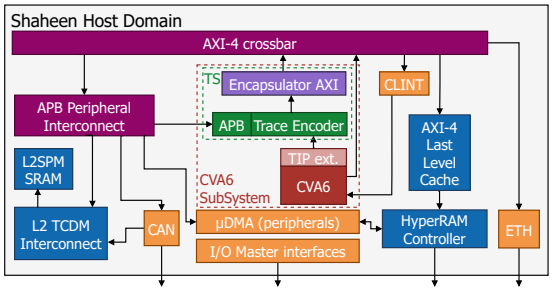
This work presents three main contributions: i) The design of a Tracing System (TS) compliant with the RISC-V E-Trace specification [2]; ii) Its integration into a modern RISC-V edge platform based on the CVA6 core [3]; iii) The evaluation of the proposed implementation in terms of achieved compression rate and FPGA resource utilization overhead.

Architecture

Figure 1a illustrates the TE architecture. It takes as input the length of the first instruction in the block (`iaddr`), the type and length of the last instruction in the block (`itype`, `ilastsize`), the block length (`iretire`), the privilege level (`priv`), and interrupt/exception details (`tval`, `cause`). The TE consists of three main building blocks, `te_filter`, `te_priority`, and `te_packet_emitter`, which are responsible for identifying trace blocks and generating E-trace packets. Additionally, it includes three support and configuration modules, namely `te_reg`, `te_resync_counter`, and `te_branch_map`. The `te_filter` determines which information to trace. Its parameters are stored in the `te_reg` module which manages user-configurable settings, including trace enablement, operating mode selection, and filter definitions to refine trace scope, and it can be configured via an APB interface. The `te_priority` module determines which packet to issue according to E-Trace. The `te_packet_emitter` module is responsible for constructing the trace packets. It receives the packet type from the `te_priority` module and collects the required data from the `te_reg` module, as well as delayed TE inputs from registers. The inputs are delayed



(a) *Trace Encoder architecture.*



(b) *Shaheen host-domain extended with TS.*

Figure 1: *TE architecture and integration inside Shaheen*

by up to two cycles, allowing it to access them from current, previous and next cycle. Tracking discontinuities in these three states is essential to select the appropriate packet for output. The `te_branch_map` module monitors branch instructions and keeps track of their results. It uses a 31-bit register to store branch results (the branch map) and a counter. When the branch map is full, the module requests the generation of a packet to report branch information, so no branch data is lost. The `te_resync_counter` module tracks the number of packets emitted or clock cycles elapsed. When the predefined threshold is reached, it triggers a resynchronization request to the `te_priority` module.

A key feature of the TE is its ability to support processors that can retire multiple instructions per cycle. This is achieved by replicating block-specific inputs (`iaddr`, `itype`, `ilastsize`, `iretire`) and `te_filter`, `te_priority` and `te_packet_emitter` modules based on the number and type of discontinuities that can be retired by the CPU each cycle.

The E-Trace packets generated by the TE are transferred to an AXI4 encapsulator module via a valid/ready handshake. Upon receiving the payload, it generates AXI4 transactions to route trace packets through the system crossbar and transmit them over Ethernet. Figure 1b shows the integration of our design into the host domain of Shaheen [3], a state-of-the-art RISC-V platform featuring a CVA6 core for autonomous nano-drones. We extended the CVA6 interface with a Trace Interface Port (TIP) that generates TE inputs according to E-Trace [2].

Results and Conclusions

We conducted a preliminary evaluation of the TS by integrating it into Shaheen and implementing it on an FPGA emulator on the Xilinx VCU118 FPGA. We evaluated the area overhead and compression rate achieved using platform-specific benchmarks and regression tests. Figure 2 shows the area overhead introduced by the Tracing System over CVA6. The TS consumes about 9.2% of the total resources of the CVA6 subsystem (which includes both the CVA6 core and the TS) and about 10% of the area compared to the CVA6 core alone. The TIP and Encapsulator AXI consume area primarily due to the registers that implement FIFOs, which are essential for elastic buffering. In area-constrained scenarios, the TS footprint can be reduced by instantiating smaller depth FIFOs. However, decreasing the FIFO depth may result in packet loss, which could affect the completeness of the captured trace.

In addition, the modules do not have an impact on the critical path, so there is no impact on operating frequency.

Table 1 presents the compression rate achieved by the tracing system, with an average compression rate

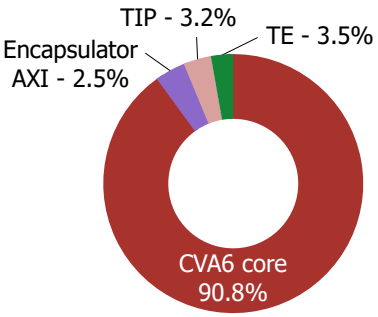


Figure 2: *Resources utilization with respect to the CVA6 subsystem*

Test name	Compression rate %
`axi_hyper_fibonacci`	99.8
`bypass_cva6_dco`	99.5
`can`	87.3
`dhrystone`	98.4
`fp16_matmul`	99.7
`fp16-vec_matmul`	99.7
`hello`	90.4
`kmeans`	95.0
`l1_test`	99.7
`llc_spm_test`	87.6
`mbox_test`	91.7
`mm`	99.7
`sb_macl_444`	99.7
`sb_macl_844`	99.7
`timer`	85.2
Average	95.1

Table 1: *Compression rate for each test*

of 95.1% compared to tracing each instruction with its full-opcode allocation.

We developed a RV TS and integrated it into an edge platform based on CVA6 called Shaheen. For future work, we plan to conduct more comprehensive benchmarks, enhance the TE with additional hardware features to achieve higher compression rates and advanced functional monitoring, and formally release the implementation open-source. Additionally, the development of the TD is currently ongoing.

Acknowledgements This activity has been supported by the HE EU EUPEX (g.a. 101033975), DECICE (g.a. 101092582), and DARE (g.a. 101143421) projects, as well as the Italian Research Center on High Performance Computing, Big Data, and Quantum Computing.

References

[1] Charlie Curtsinger et al. “Coz: Finding code that counts with causal profiling”. In: Proceedings of the 25th Symposium on Operating Systems Principles. 2015.

[2] Gajinder Panesar et al. “Efficient Trace for RISCV”. In: Siemens, ver. 1.1. 3-Frozen (2022).

[3] Luca Valente et al. “A Heterogeneous RISC-V Based SoC for Secure Nano-UAV Navigation”. In: IEEE Transactions on Circuits and Systems I: Regular Papers (2024).